Secure Containers in Embedded Deployments

Solutions for containers in embedded

Stefano Stabellini @stabellinist

The Problem

The problem

Package applications for the target

Contain all dependencies

Easy to update, Independent lifecycle

Run applications on the target

Run in isolation

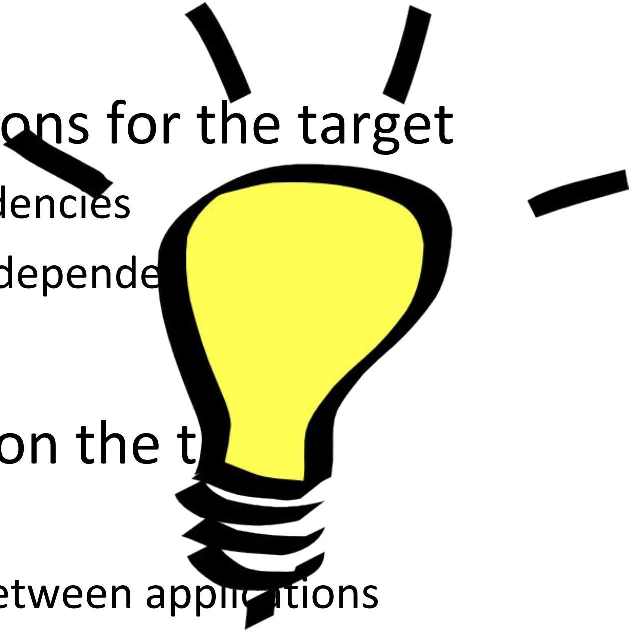
No interference between applications

The problem

Package applications for the target

Contain all dependencies

Easy to update, Independe



Run applications on the t

Run in isolation

No interference between applications

The problem

Package applications for the target

Contain all dependencies

Easy to update, I

Run applications

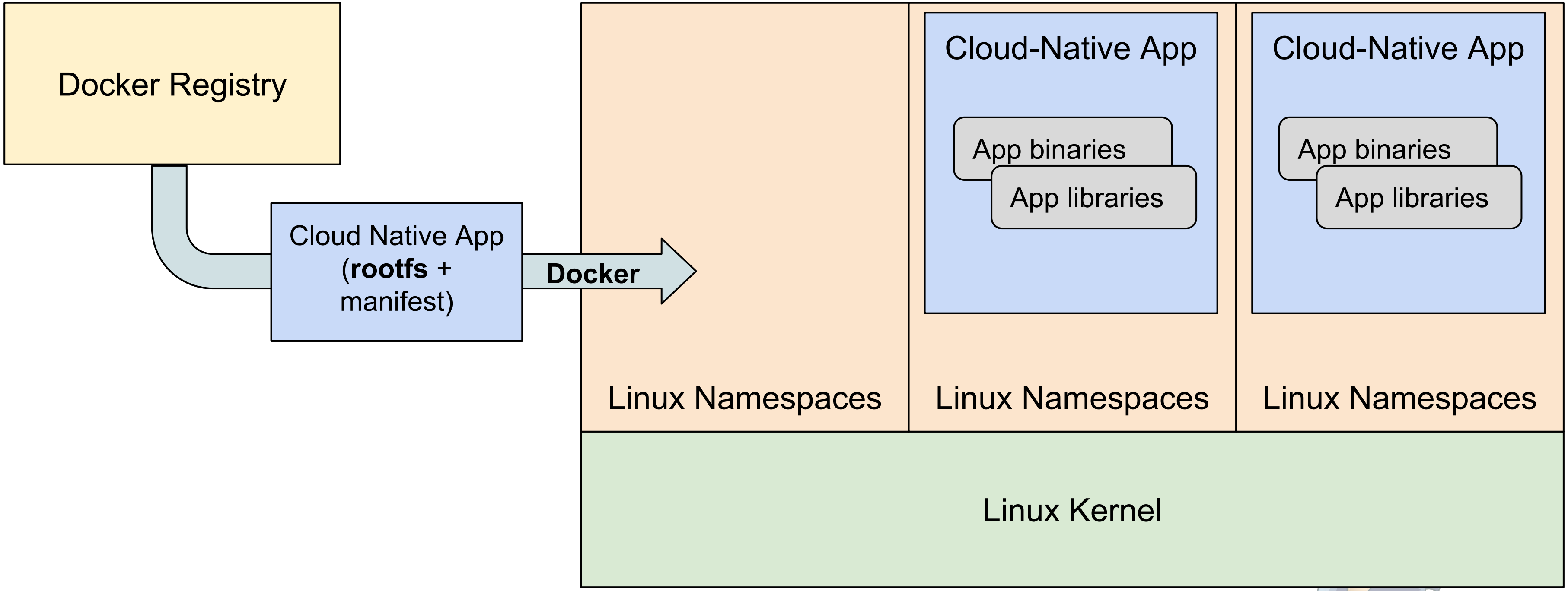
Run in isolation

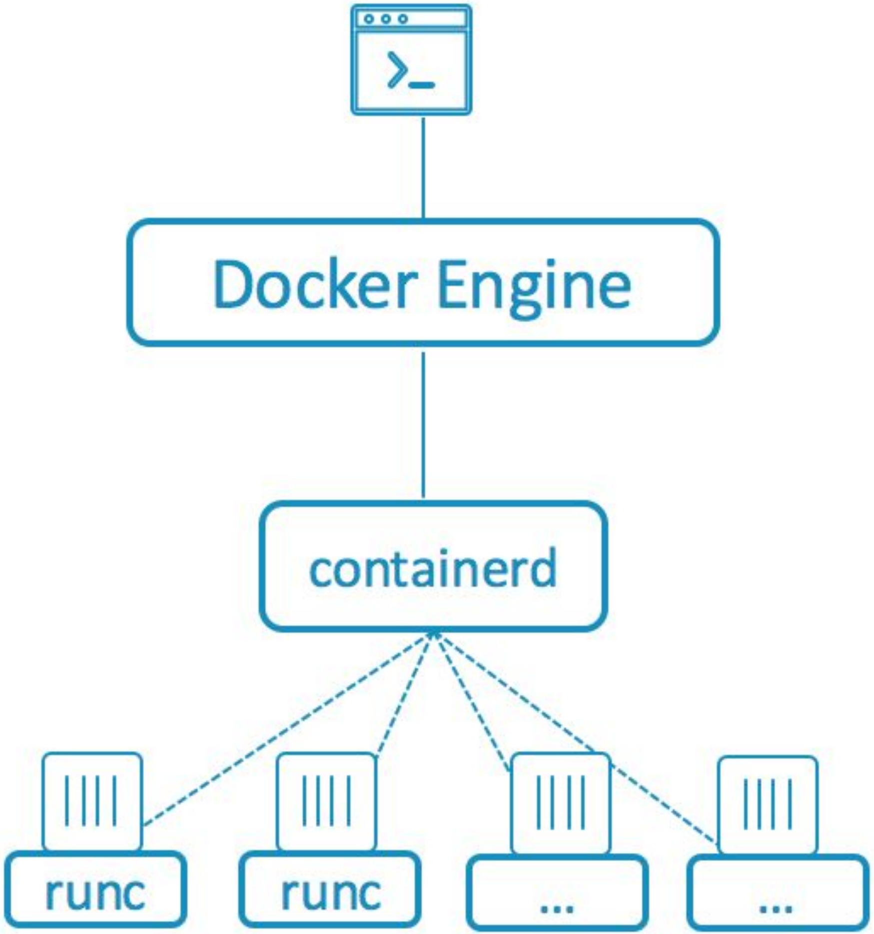
No interference between applications

Packaging vs. Runtime

OCI Image Spec vs. OCI Runtime Spec

Containers != Linux Namespaces





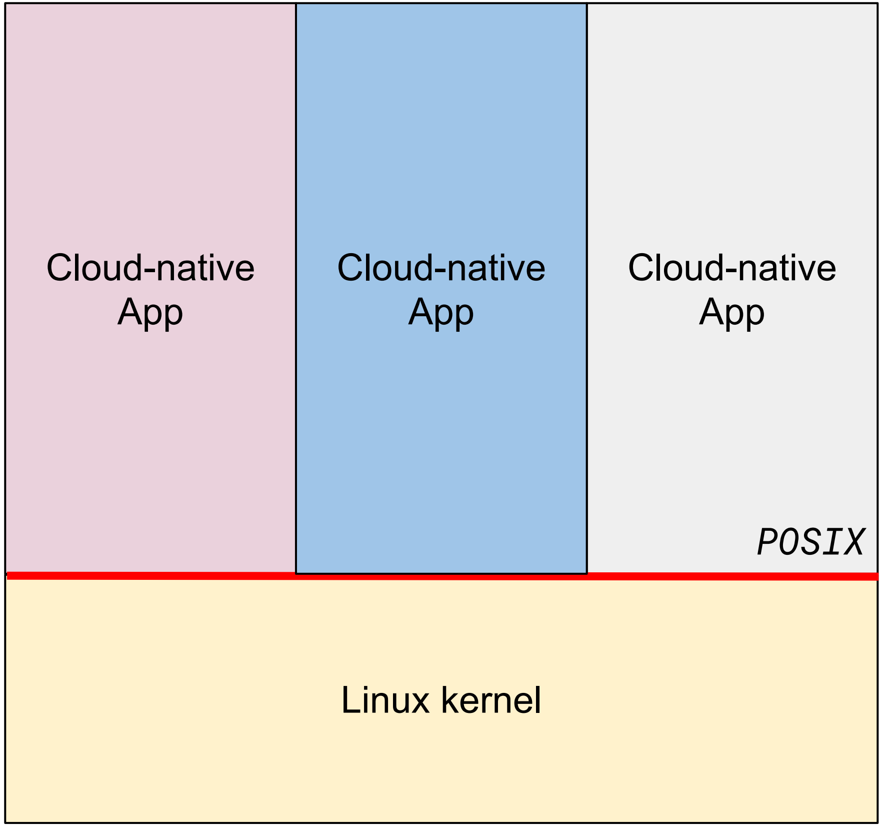
Same Docker UI and commands

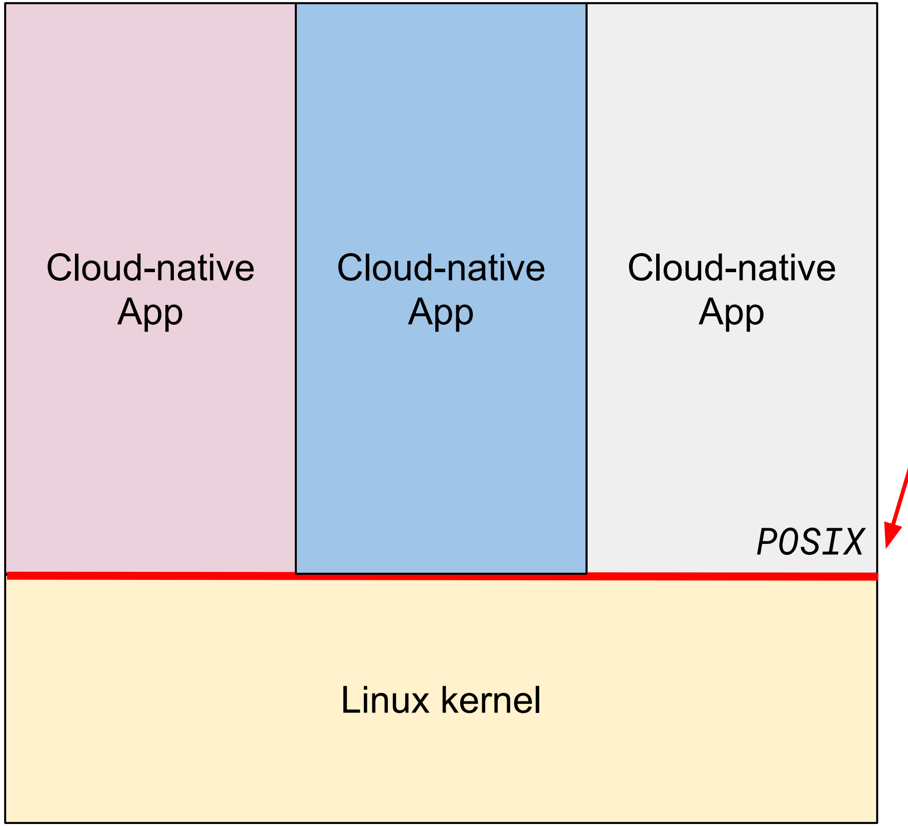
User interacts with the Docker Engine

Engine communicates with containerd

containerd spins up runc or other OCI compliant runtime to run containers

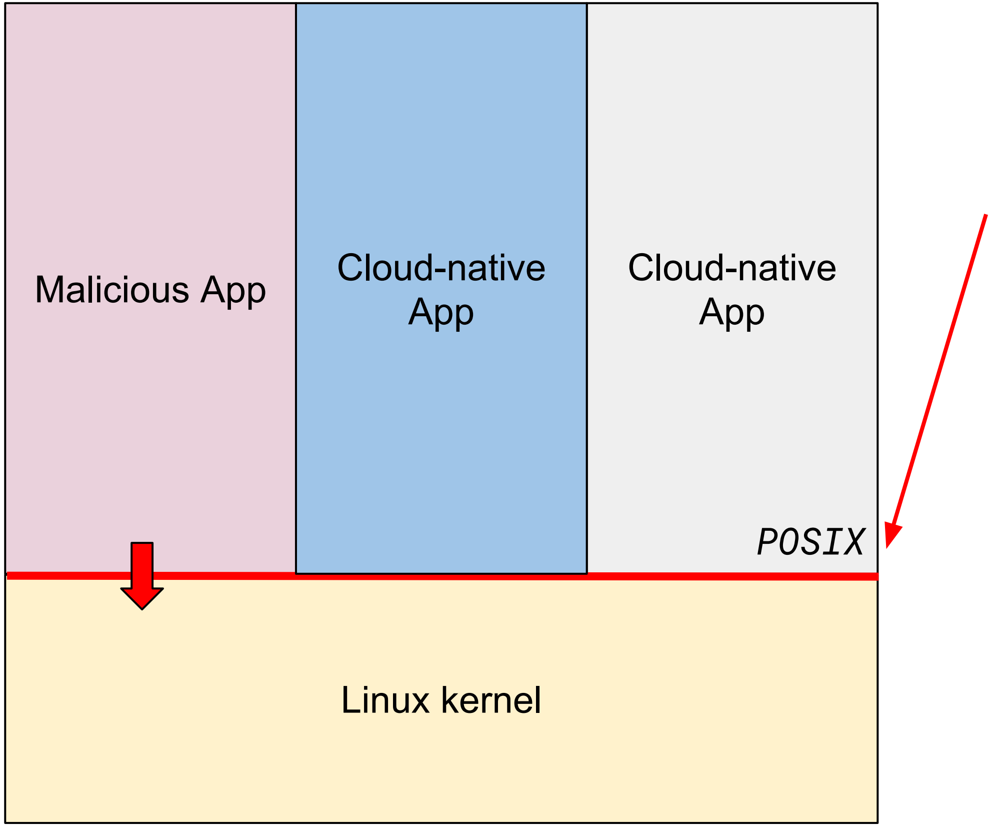
The problem with Linux namespaces


Cloud-native App
Cloud-native App
Cloud-native App
POSIX
Linux kernel



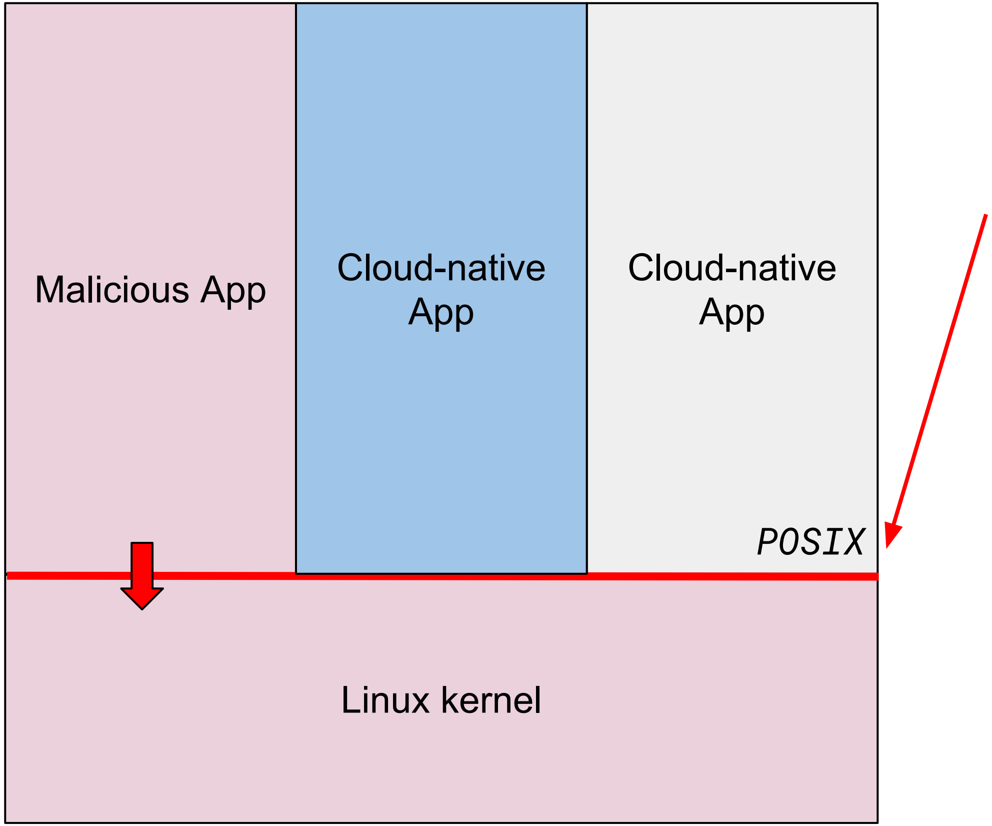
Large surface of attack

On average, 3 privilege escalation vulnerabilities per Linux release!



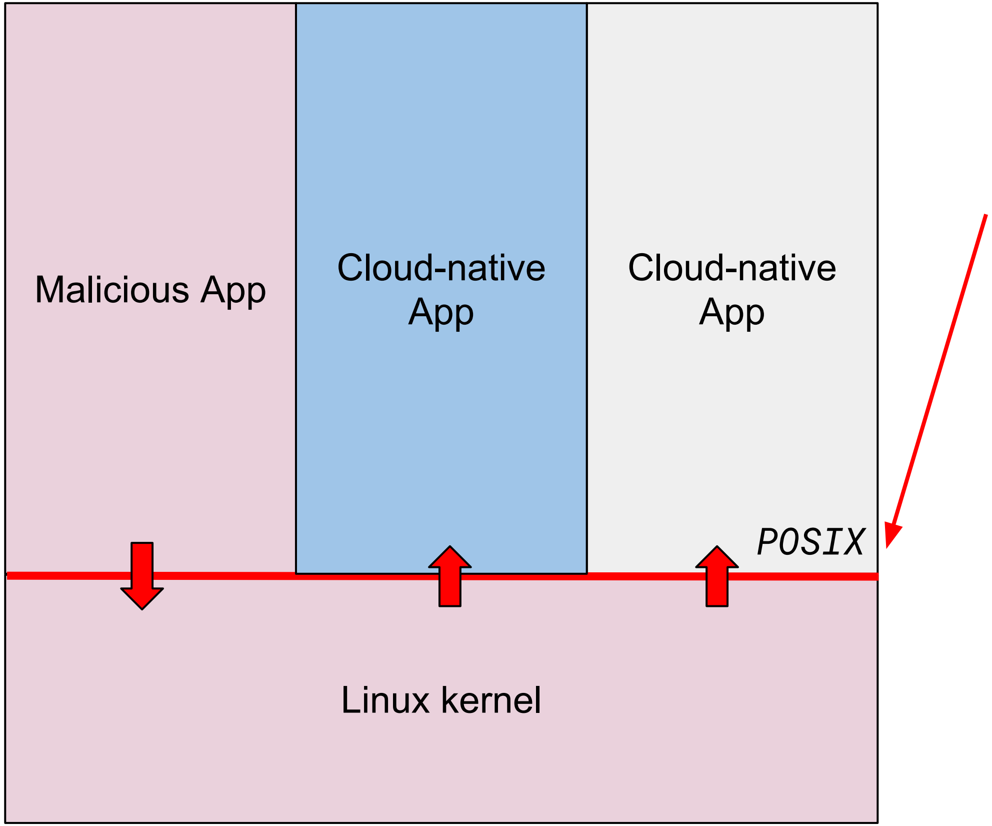
Large surface of attack

On average, 3 privilege escalation vulnerabilities per Linux release!



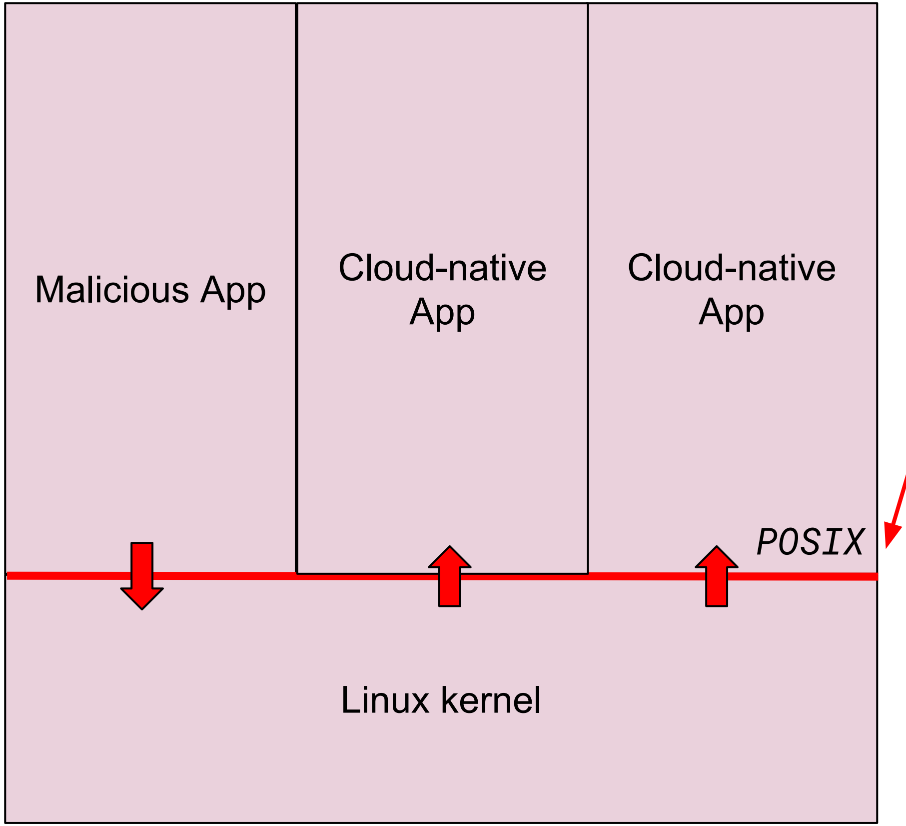
Large surface of attack

On average, 3 privilege escalation vulnerabilities per Linux release!



Large surface of attack

On average, 3 privilege escalation vulnerabilities per Linux release!



Large surface of attack

On average, 3 privilege escalation vulnerabilities per Linux release!

Security hardening techniques

From “Understanding and Hardening Linux Containers” by NCC Group:

- Run unprivileged containers (user namespaces, root capability, dropping)
- Apply a Mandatory Access Control system, such as SELinux
- Build a custom kernel binary with as few modules as possible
- Apply sysctl hardening
- Apply disk and storage limits
- Control device access and limit resource usage with cgroups
- Drop any capabilities which are not required for the application within the container

[...]

Security hardening techniques

[...]

- Use custom mount options to increase defense in depth
- Apply GRSecurity and PAX patches to Linux
- Reduce Linux attack surface with Seccomp-bpf
- Isolate containers based on trust and exposure
- Logging, auditing and monitoring is important for container deployment
- **Use hardware virtualization along application trust zones**

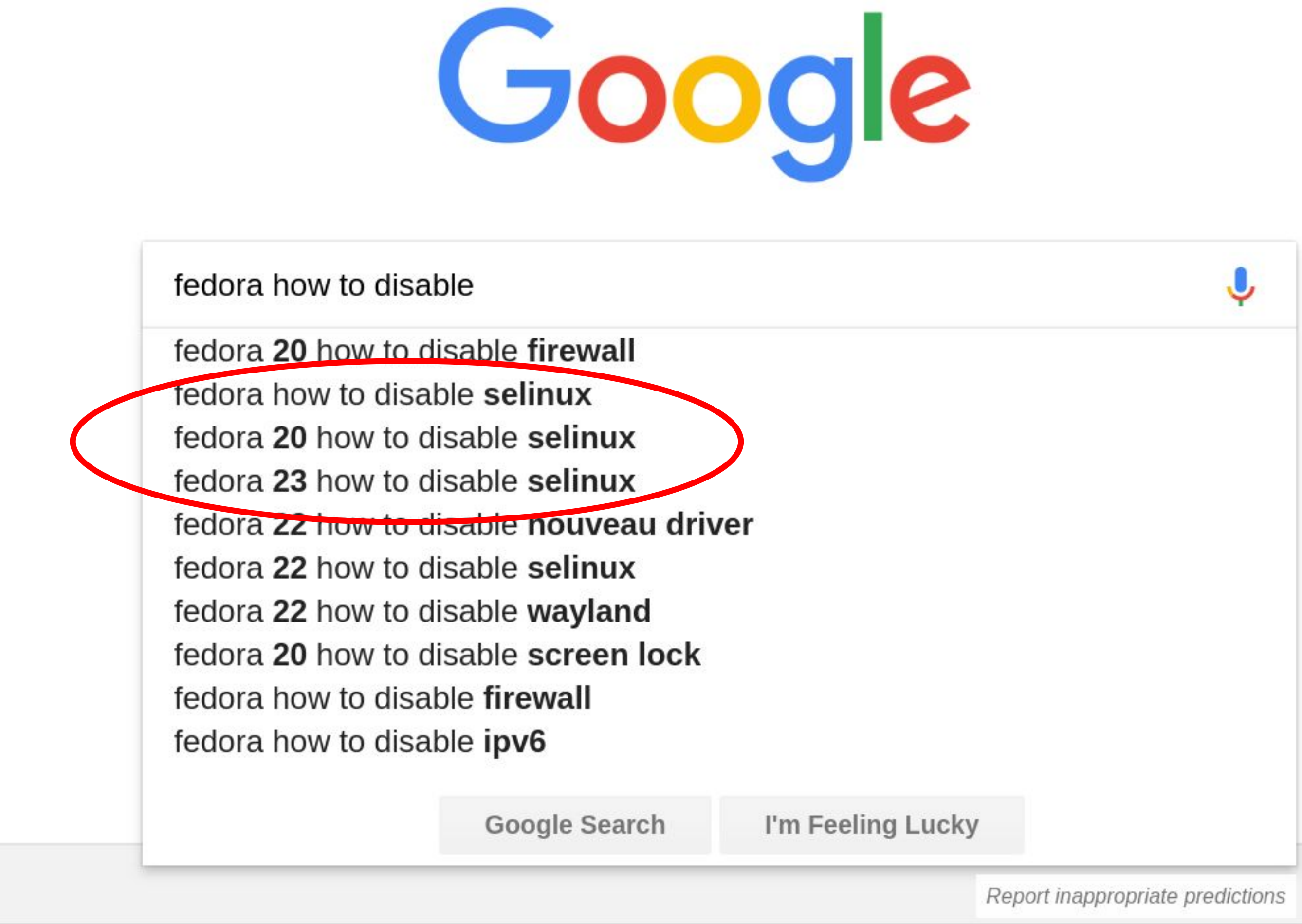
Security hardening techniques

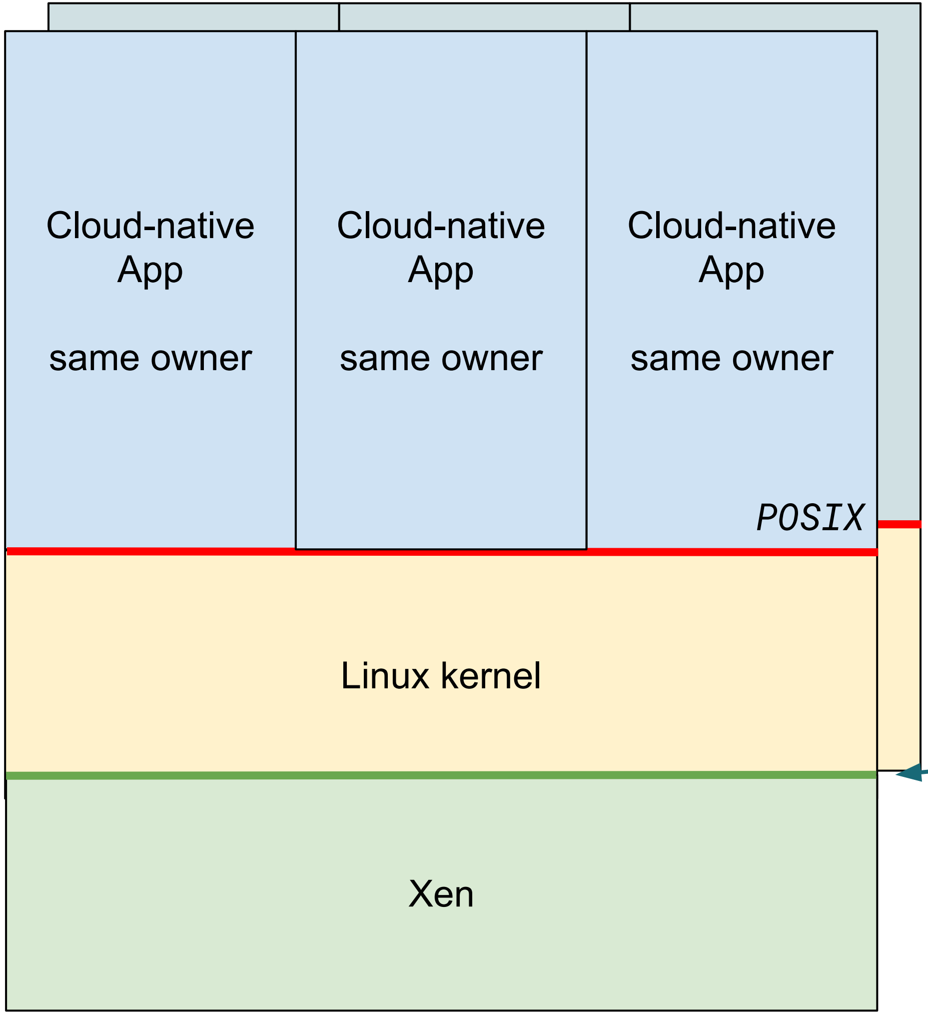
Securing Linux namespaces is ***possible*** but ***very difficult***

It requires specific knowledge of the cloud-native app

Auditing and monitoring should be performed everywhere

Using ***virtualization*** for isolation is still ***recommended***


Google
fedora how to disable
fedora 20 how to disable firewall
fedora how to disable selinux
fedora 20 how to disable selinux
fedora 23 how to disable selinux
fedora 22 how to disable nouveau driver
fedora 22 how to disable selinux
fedora 22 how to disable wayland
fedora 20 how to disable screen lock
fedora how to disable firewall
fedora how to disable ipv6
Google Search
I'm Feeling Lucky
Report inappropriate predictions



- No multi-tenancy
- Only run cloud-native apps from the same user on the same host
- Use VMs (or bare-metal) as security boundary
- Need to handle both VMs provisioning and Cloud-Native app provisioning

Virtual interface, on average:

Xen PV: 1 priv escalation vuln / year
KVM: 4 priv escalation vuln / year

Linux Namespaces: very embedded problems

Multi-tenancy is not supported

Mixed-criticality workloads are not supported

Limits on resources utilization hard to enforce

Real-time support is difficult

Certifications are very difficult

Linux Namespaces: very embedded problems

Multi-tenancy is not supported

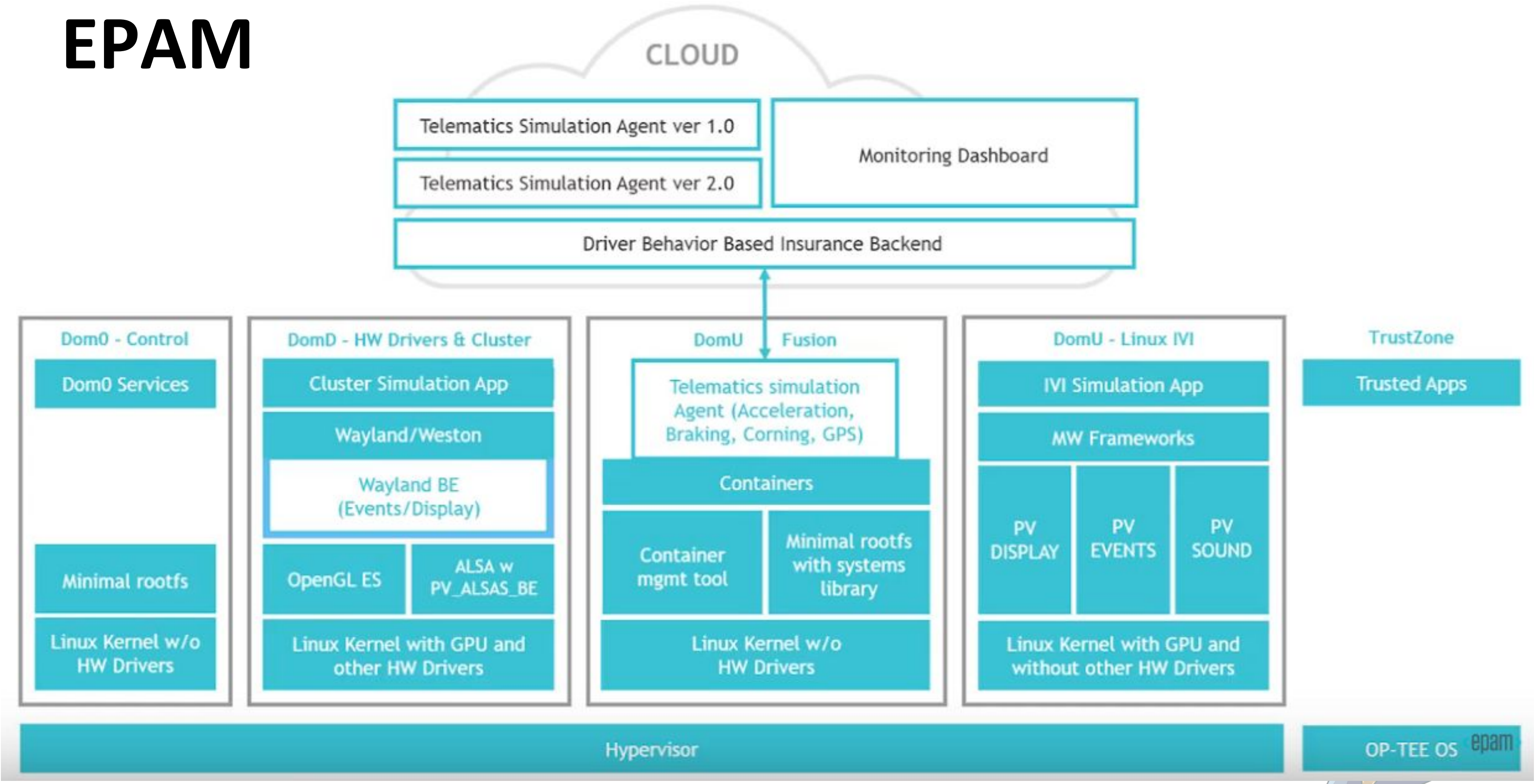
Mixed-criticality workloads are not supported

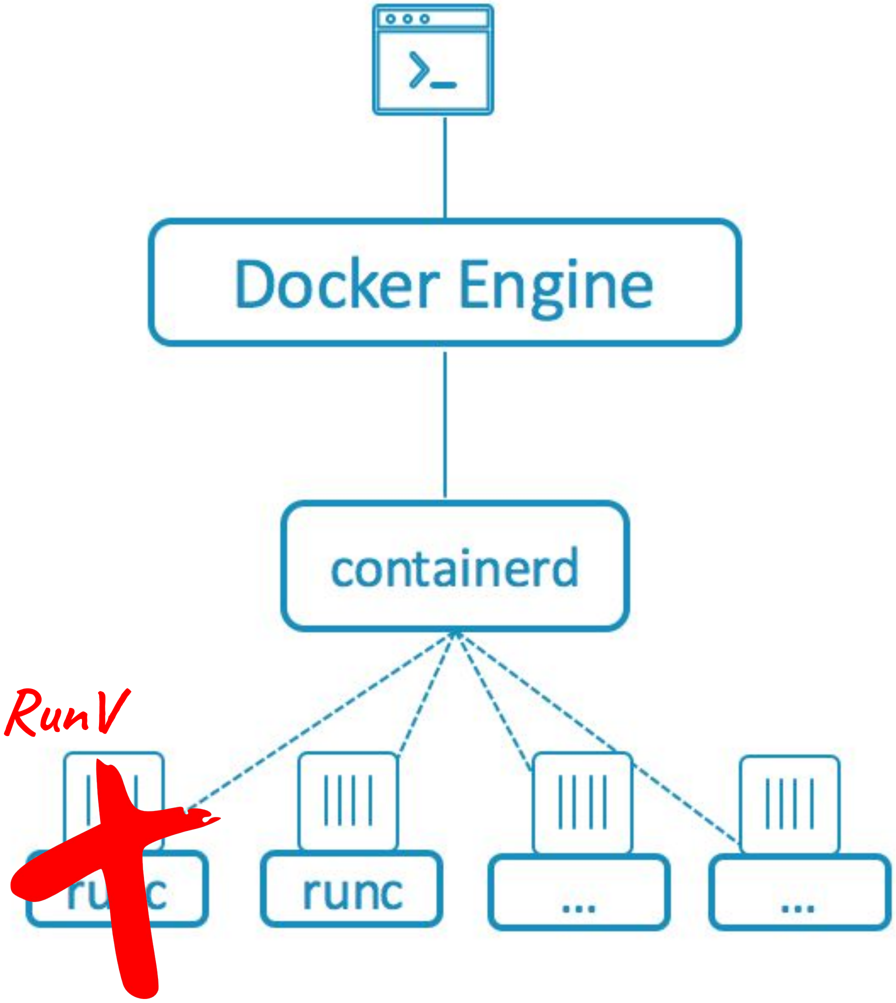
Limits on resources utilization hard to enforce

Real-time support is difficult

Certifications are very difficult

EPAM


CLOUD
Telematics Simulation Agent ver 1.0
Telematics Simulation Agent ver 2.0
Monitoring Dashboard
Driver Behavior Based Insurance Backend
Dom0 - Control
Dom0 Services
Minimal rootfs
Linux Kernel w/o HW Drivers
DomD - HW Drivers & Cluster
Cluster Simulation App
Wayland/Weston
Wayland BE (Events/Display)
OpenGL ES
ALSA w PV_ALSAS_BE
Linux Kernel with GPU and other HW Drivers
DomU Fusion
Telematics simulation Agent (Acceleration, Braking, Corning, GPS)
Containers
Container mgmt tool
Minimal rootfs with systems library
Linux Kernel w/o HW Drivers
DomU - Linux IVI
IVI Simulation App
MW Frameworks
PV DISPLAY
PV EVENTS
PV SOUND
Linux Kernel with GPU and without other HW Drivers
TrustZone
Trusted Apps
Hypervisor
OP-TEE OS
epam



Same Docker UI and commands

User interacts with the Docker Engine

Engine communicates with containerd

containerd spins up runc or other OCI compliant runtime to run containers

Virtualization as container runtime

Virtualization

- Security, Isolation and Partitioning
- Multi-tenancy
- Mixed-criticality workloads
- “Componentization”
- Resilience
- Hardware access to applications
- Real-time support

Hypervisors in Embedded != Cloud

Different requirements:

- **small codebase (safety, certifications)**
- **real time schedulers**
- low, deterministic irq latency
- short boot times
- small footprint
- non-PCI device assignment
- driver domains
- co-processor virtualization

Hypervisors in Embedded != Cloud

Different requirements:

- **small codebase (safety, certifications)**
- **real time schedulers**
- low, deterministic irq latency
- short boot times
- small footprint
- non-PCI device assignment
- driver domains
- co-processor virtualization

Xen® Project On ARM

Xen Project

The hypervisor with a micro-kernel design

Extensive feature-set, highly customizable

real time, device passthrough (x86, ARM32, ARM64), wide hardware support, PV drivers

Small codebase < 60K   supports Kconfig

Real-time support out of the box: real time schedulers, pinning

Xen on ARM: A lean and simple architecture

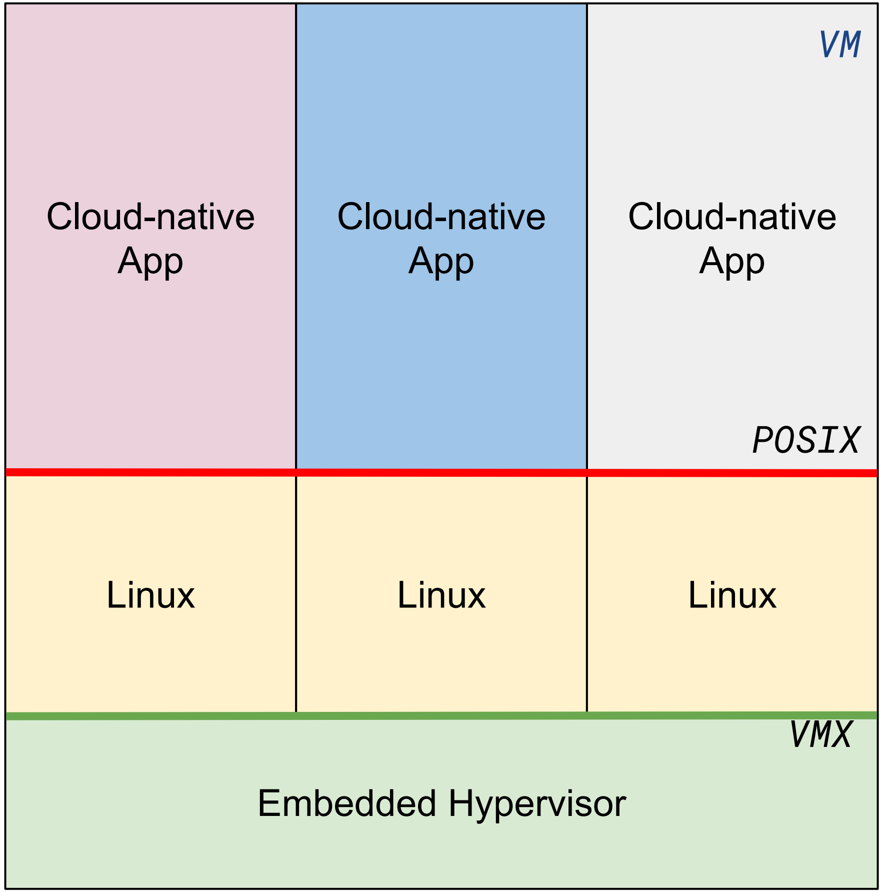
No cruft, No emulation, No QEMU; Small attack surface; One type of guest

PVH guests already available on x86; PVH-only Xen in development

Transparent Security Process

Yes but,
Does it run containers?

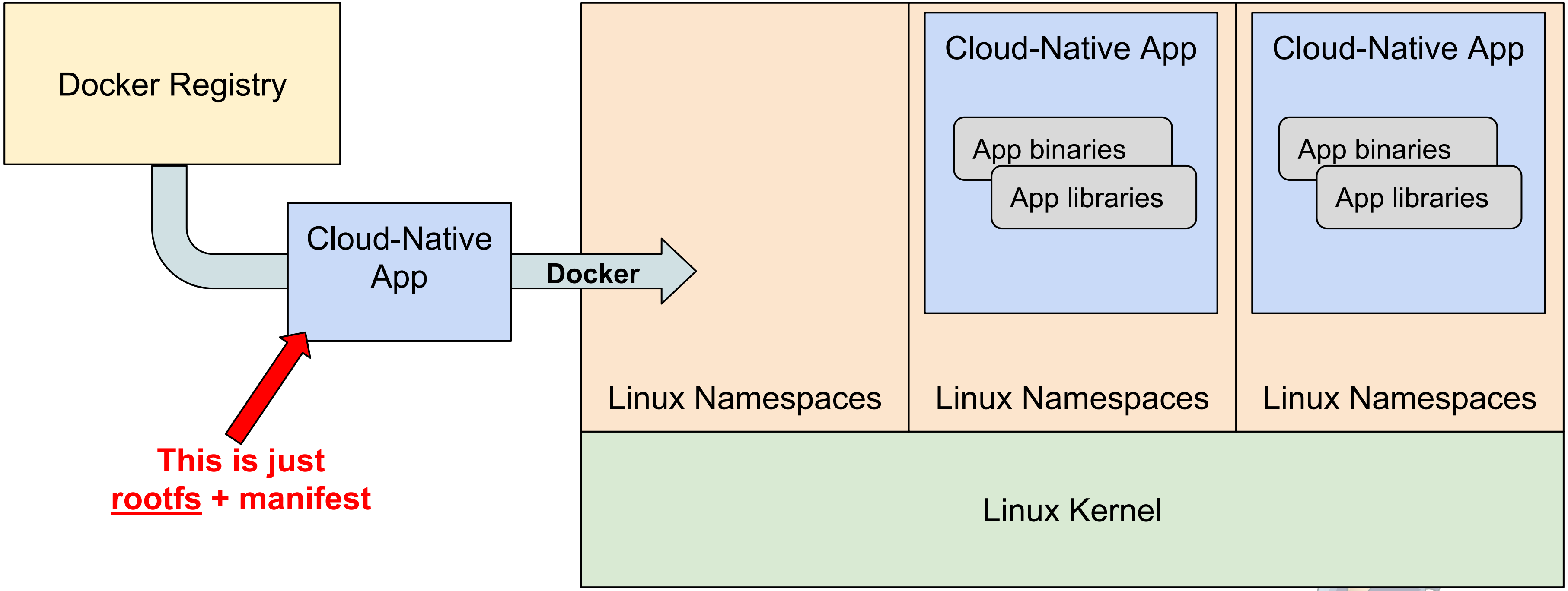
Xen as container runtime



- 1 container app <--> 1 VM
- Secure by default
- Mix and match traditional VMs and container apps on a single platform
- Support mixed criticality workloads
- Support real time apps
- Support device assignment

How do we do it?

Containers != Linux Namespaces



Containers for packaging, Xen for runtime

1. Fully static use-cases: use containers as a packaging format

   extract the rootfs, run each container as Virtual Machine manually

   see **singularity** http://singularity.lbl.gov/

2. Run containers as VMs automatically with **rkt and stage1-xen**

   strong isolation

   support multi-tenancy and mixed-criticality workloads

   support real time requirements

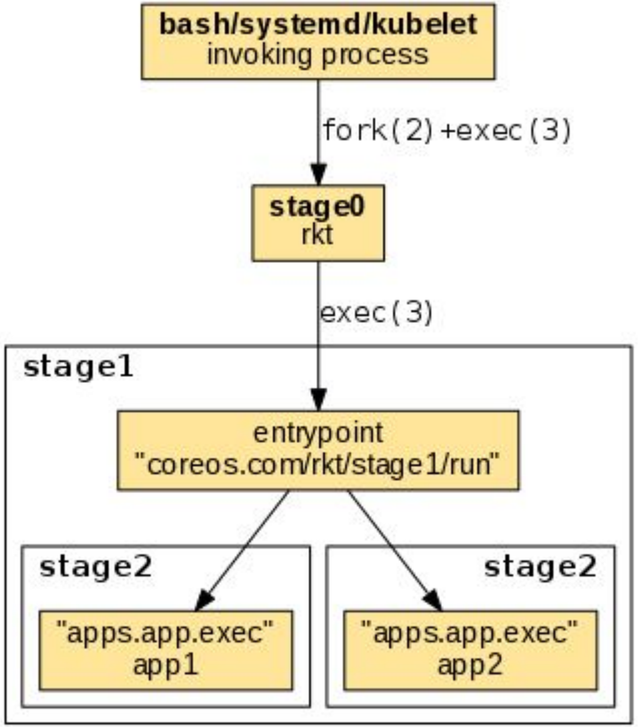
   also see RunV, Kata Containers, KubeVirt, Virtlet

CoreOS rkt

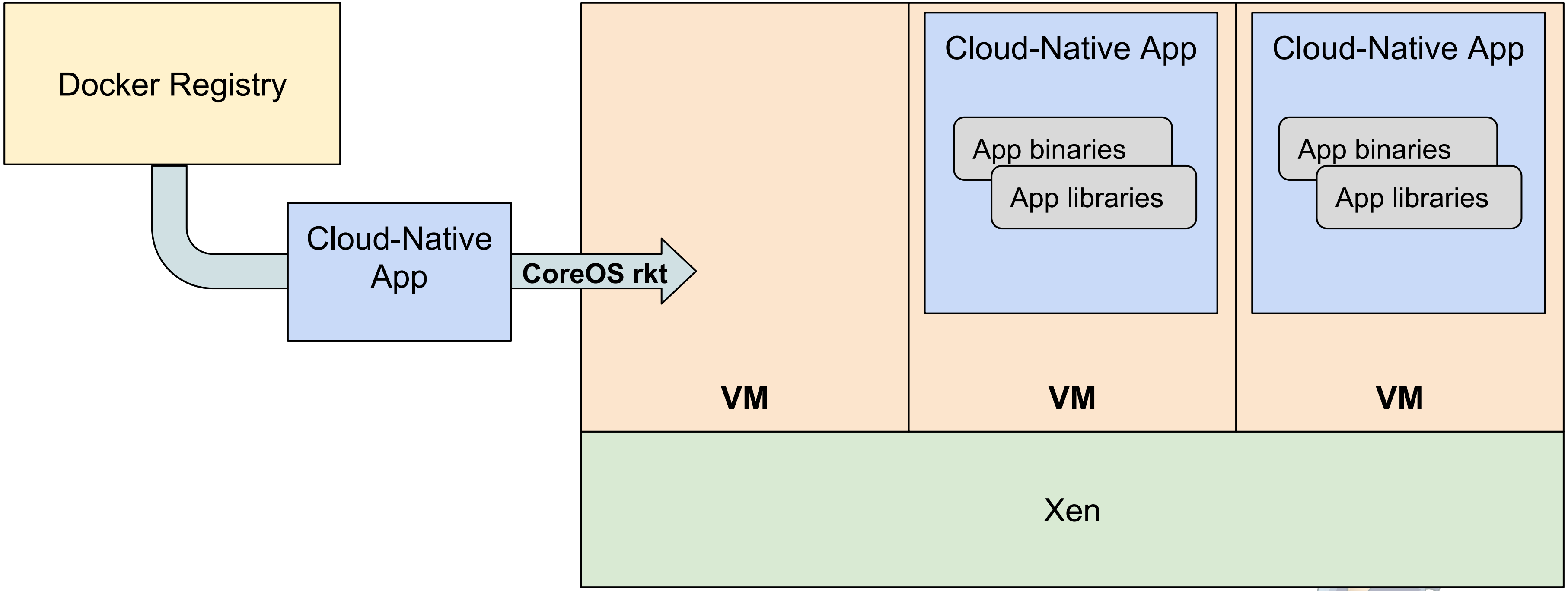
rkt

A security-minded, standards-based container engine

CoreOS rkt


bash/systemd/kubelet
invoking process
fork(2)+exec(3)
stage0
rkt
exec(3)
stage1
entrypoint
"coreos.com/rkt/stage1/run"
stage2
"apps.app.exec"
app1
stage2
"apps.app.exec"
app2

Introducing stage1-xen



Stage1-xen: design

- ACI format = tarball + manifest
- well defined entry points
- based on xl and 9pfs
- written in bash and golang
- multiple networking models (bridge, nat, pvcalls)

PVCalls

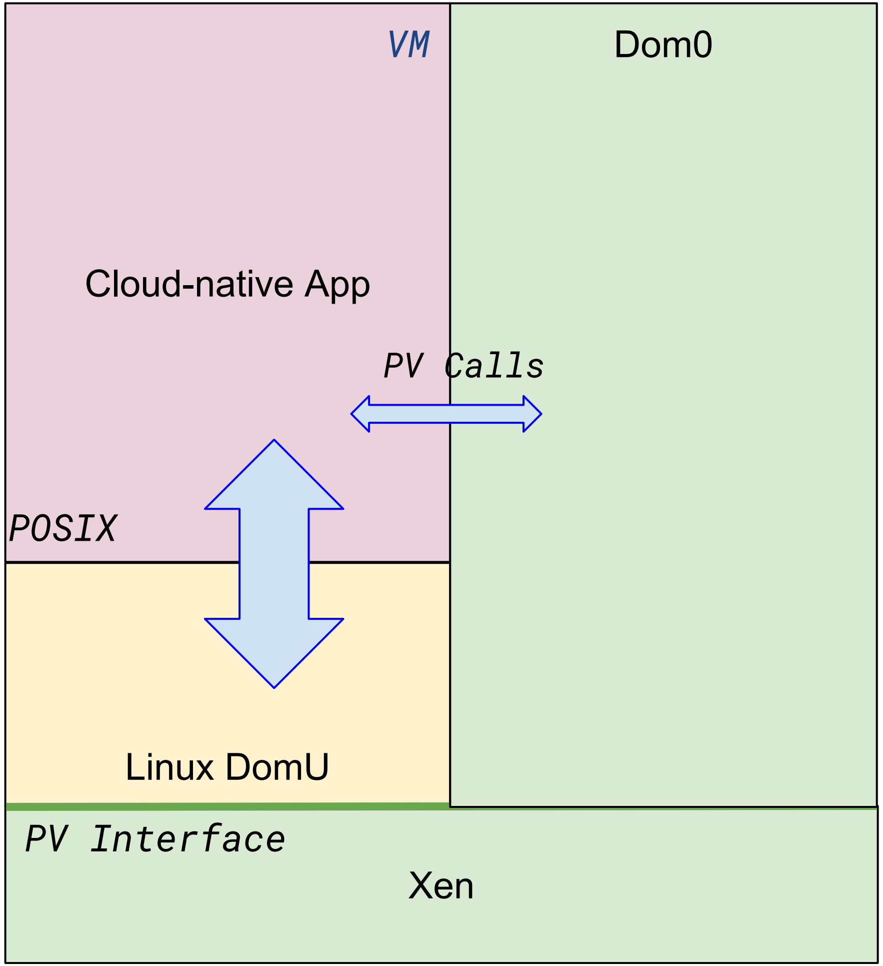
PV Calls

Only support POSIX apps -> Virtualize at the POSIX level

Few selected POSIX calls are sent to Dom0

- it’s the right abstraction layer for cloud-native apps
- monitoring apps becomes easy and cheap
  - monitor network and filesystem access
  - easy to identify changes in access patterns
- very good performance

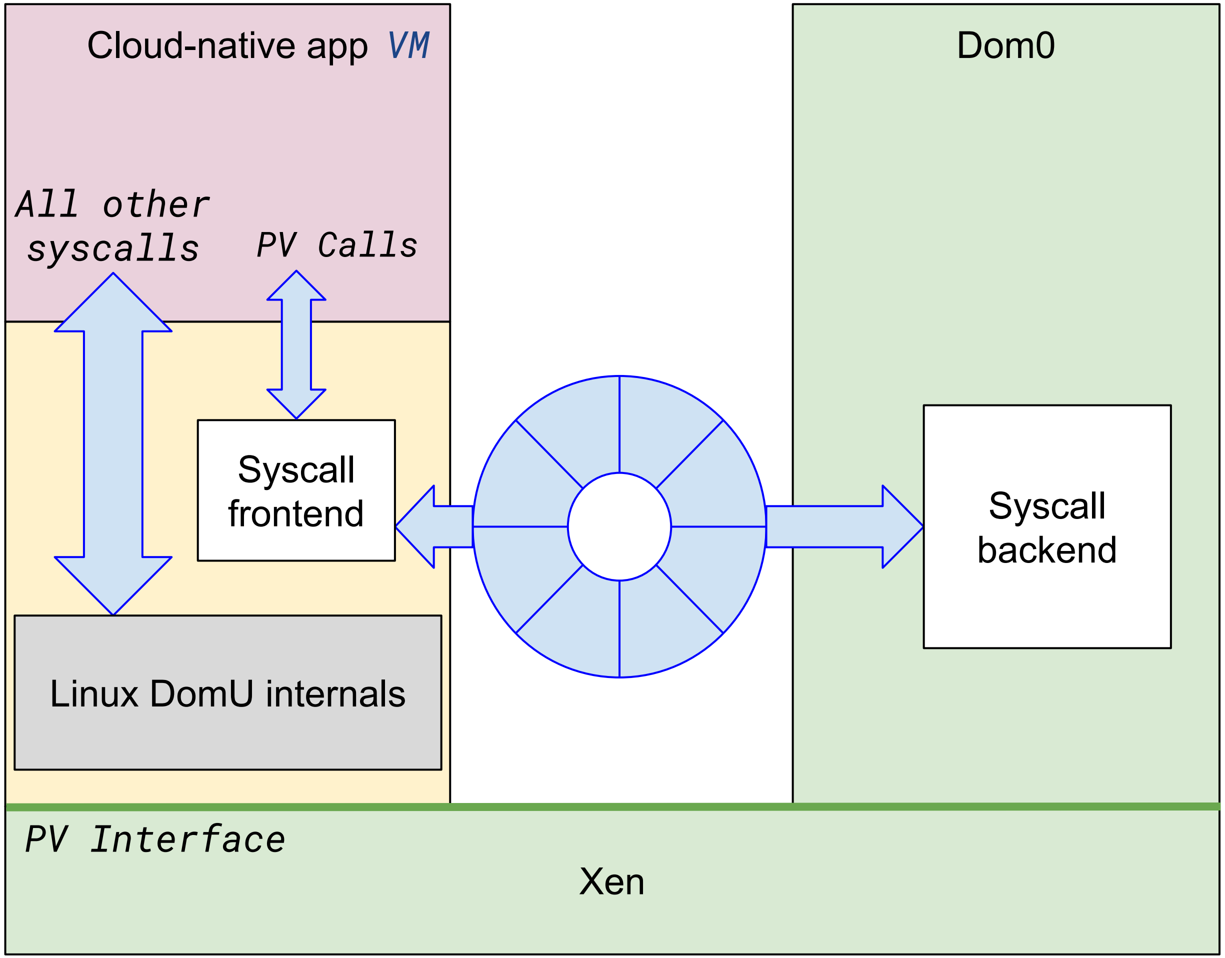
PV Calls



Each app is run in a small separate Xen VM for *isolation*.

POSIX calls are confined within the VM, *“emulated”* by the guest kernel.

Few selected syscalls are handled securely by Dom0 (*filesystem* and *socket* syscalls primarily).



Cloud-native app VM
Dom0
All other syscalls
PV Calls
Syscall frontend
Syscall backend
Linux DomU internals
PV Interface
Xen

PV Calls for networking

- Ports opened in a VM, are opened on the host
- A great match for containers engines
- Bind VM network calls to different dom0 network namespaces
- Zero-conf networking in VMs
  - no need for a bridge in dom0
  - works with wireless networks, VPNs, any other special configurations in Dom0

Considerations on Meltdown

Meltdown

Linux (x86 and ARM) is affected

Xen on ARM Virtual Machines are unaffected

PVH and HVM Virtual Machines on x86 are unaffected

PV Virtual Machines on x86 are affected, Xen was fixed

Performance: Meltdown aftermath

Intel NUC 5i5MYHE

2 Intel Core i5-5300U CPU @ 2.30GHz

4GB of RAM

Xen 4.11-unstable CS 52ba201362aab4b09d44bcca67967c1053721ac2

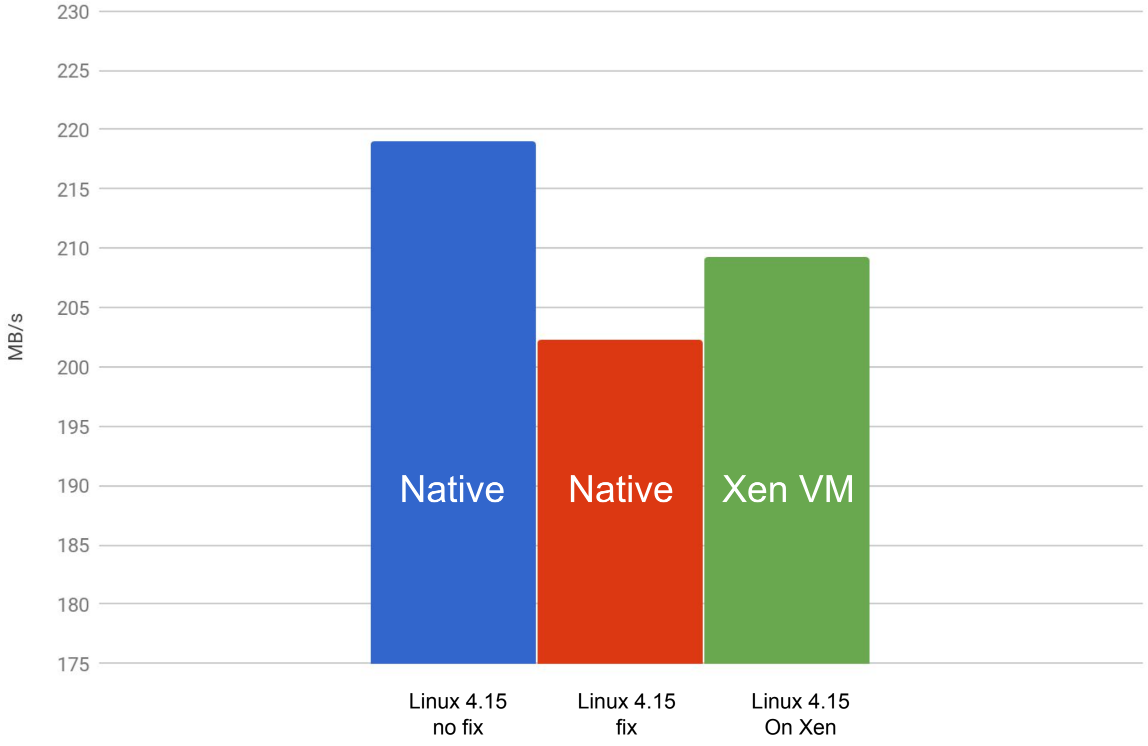
Linux 4.15 with and without CONFIG_PAGE_TABLE_ISOLATION

Dom0: 1.4G RAM, 2 vcpu

DomU / Native: 2G RAM, 2 vcpus

Performance: Meltdown aftermath

CompileBench, Higher is Better



Conclusions

Containers are a great packaging format

Linux namespaces are not suitable for all use-cases

Virtualization offers a secure-by-default runtime environment

Watch out for announcements at

blog.xenproject.org

and

www.linuxfoundation.org

in the next few months!

Demo

Stefano Stabellini
sstabellini AT kernel.org
twitter.com/stabellinist